N76-23323

An INTEL 8080 Cross Assembler for the Modcomp II Minicomputer

M. W. Sievers
Communications Systems Research Section

The flexibility of Modcomp's macro assembler has been exploited to implement an INTEL 8080 cross assembler. This simple implementation is very powerful, allowing, for example, macro definitions, and declaration of common and external labels. The cross assembler may be executed on any Modcomp II minicomputer.

I. Introduction

A macro assembler is a special type of assembler that permits the definition of prototype constructs or macros. Each macro prototype is labeled and may consist of other macro references, argument paraforms and/or assembly language code. A macro is referenced by its label; each reference is usually followed by an argument list. During pass 1 of the macro assembler, macro prototypes are placed in a prototype table. Each time the macro assembler recognizes a macro label in the op code field of a source statement, it fetches the prototype for that macro from the table and replaces argument paraforms with arguments from the argument list. The resultant construct is then inserted into the source after the statement in which it was referenced. After completing pass 1, the usual second assembler pass is invoked which produces a complete binary object file.

The flexibility of macro assemblers can be exploited to generate cross assemblers for virtually any machine with minimal effort. A cross assembler can be implemented by defining a set of macro prototypes whose labels correspond to the mneumonics in the target assembly language. Once these prototypes have been defined, assembly directives for the target machine may be assembled by the host machine's macro assembler.

Binary object produced by the cross assembler described above is not necessarily directly transferable to the target machine. For example, differences may arise in word length or byte ordering. These differences can be resolved by a loader routine. The complexity of the loader naturally depends on the complexity of the differences which it must resolve. It can be expected however, that a loader routine will in general be far simpler to write than an assembler that generates directly transferable object.

This paper will describe an INTEL 8080 cross assembler and loader that executes on a Modcomp II minicomputer. The cross assembler is very flexible

allowing, for example, the definition of macro prototypes, and the declaration of common and external labels. If desired by the user, the loader can write binary output onto paper or magnetic tape for easy transport to an 8080 microcomputer.

II. 8080 Assembler

An 8080 cross assembler has been implemented as described above by defining a collection of macro prototypes written in Modcomp's macro assembly language (ref. section VIII of Modcomp's Assembler Reference Manual, TM16094). These macros are labeled with 8080 mneumonics and collectively stored on disk under the label ASM8080. ASM8080 macros are inserted into an 8080 source program at assembly time via the INSERT directive (ref. section IV, TM16094, for a discussion of the INSERT directive). Once ASM8080 has been inserted, Modcomp's macro assembler can assemble 8080 source code.

Four macro prototypes found in ASM8080 are shown below. The first three macros define, respectively, one-, two-, and three-byte 8080 instructions. The fourth macro is a special address macro which is referenced by triple byte instruction macros (see below).

```
*SINGLE BYTE MACRO
HLT MAC
    DFC #76
    EMP
*DOUBLE BYTE MACRO
IN MAC
    DFC #DB
    DFC %1
    EMP
*TRIPLE BYTE MACRO
CALL MAC
    DFC #CD
    ADDR %1,%2
    EMP
*SPECIAL ADDRESS MACRO
ADDR MAC
    IFM %2,A
    DFC %1,%2
    EXM
A AOP
    DFC #FFFF, %1
    EMP
```

The single byte macro above is typical of all single byte 8080 instruction macros. It consists of a label, in this case HLT (HALT), which corresponds to an 8080 mneumonic, and a DFC (DEFINE CONSTANT) statement. The DFC defines a constant equal to the value of the operation code for the mneumonic.

The macro for the IN (INPUT) instruction shown above is a typical double byte instruction macro. Two DFC statements define, respectively, the operation code for the IN mneumonic and an argument paraform for its operand. As mentioned previously, the argument paraform is replaced by an actual argument at assembly time.

The CALL (CALL SUBROUTINE) macro is a typical three-byte instruction macro. In addition to defining an operation code constant, the address macro ADDR is referenced. ADDR examines the address field of the 8080 source statement. If the address field contains two address bytes, ADDR creates a DFC that defines the values of these bytes. If the address field contains a single address word, ADDR creates a DFC which defines a hexadecimal FFFF and the address word. The hexadecimal FFFF is a flag recognized by the loader which signals it to exchange the bytes in the address.

The 8080 cross assembler is a subset of Modcomp's macro assembler and as such contains the macro assembler's inherent flexibility and limitations. It is permissible therefore to use any of the macro assembler's pseudo operations, such as ORG (ORIGIN), COM (COMMON), RES (RESERVE), DFC (DEFINE CONSTANT), EXT (EXTERNAL), and INT (INTERNAL) to specify program origin, define blocks of common, reserve areas in core and prepare subroutines for subsequent storage in a subroutine library. The reader should refer to Section IV of Modcomp's Assembler Reference Manual for a discussion of these pseudo directives and rules governing their use. Note that Modcomp macro assembler restrictions require that the 8080 source format resemble Figure 1.

A sample cross assembly is shown in Figure 2. As already mentioned, 8080 assembly source statements are constrained to conform to all rules and limitations governing writing in Modcomp's macro assembly language. In addition to these requirements, there are the following nuances particular to this 8080 cross assembler:

(1) When using 8080 assembly directives that require a register pair, it is necessary to include both register names in the operand field. For example, PUSH B is not acceptable; PUSH BC must be used.

(2) A special macro called REGDEF is included in the ASM8080 prototype collection. This macro is used to define the value of 8080 registers and register pairs via EQUATE statements. It must be referenced in the 8080 source after all macro prototypes and common definitions but prior to any reference to 8080 registers or register pairs (ref. Figure 1).

(3) Since the register names (A, B, C, D, E, H, L, M) and register pairs (BC, DE, HL) are defined by EQUATE statements in REGDEF, register names and register pair names may not be used as labels.

(4) The dollar sign ($), when used in the operand field of a statement, refers to the current contents of the program counter plus 1.

III. Loader

The purpose of the loader is to compress binary files produced by the cross assembler and to reorder address bytes to be in the order expected by the 8080. The loader may be executed only after all addresses and common have been resolved. Modcomp's link editor should be used as required to perform the resolution function.

The loader is catalogued as a background overlay under the alias LDR. It reads data from the file assigned to BI and writes to the file assigned to BO. Since BI and BO may be assigned to any valid Modcomp file, it is possible for example, to read BI from disk and write BO onto paper tape. This facilitates producing transportable binary object. The BI and BO files must be assigned prior to executing the loader.

The loader recognizes special function codes inserted into the binary output of a cross assembly by the macro assembler. These codes inform the loader, for example, to reserve an area in core and initialize that area to a given value, or exchange the bytes in a word. Additionally, the loader recognizes the codes for origin directive and end of object. When one of these codes is found, control is passed to a routine in the loader that handles the particular code. With the exception of the code for exchanging bytes, the function codes are discussed in an appendix of the Modcomp Macro Assembler Reference Manual.

The other function performed by the loader is that of compressing the object file by removing extraneous zero high-order bytes. These bytes are inserted by the macro assembler because it expects to assemble sixteen-bit Modcomp words and not eight-bit 8080 words. Refer to the sample assembly in Figure 2.

Figures 3 through 6 illustrate the flowchart for the loader. The main routine (Figure 3) recognizes the special function codes described above and determines what routine to execute. Subroutine GTRCRD (Figure 4) reads records from the BI file into a core buffer called IBUFF. Subroutine GETWORD (Figure 4) fetches words out of IBUFF and stores them in buffer INEXT. If GETWORD is asked to fetch a word beyond the last word in IBUFF, it calls GTRCRD. Subroutine SCAN (Figures 5 and 6) scans portions of the input buffer for the exchange byte code. If that code is found, SCAN replaces the code with the low-order byte of the next word and shifts the high-order byte of the next word into its low-order byte. Subroutine COMPRESS (Figure 6) examines portions of the input buffer for extraneous zero high-order bytes. If one is found, COMPRESS shifts the next non-extraneous byte into it, thereby removing it. Finally, subroutine PUTWORD (Fig. 5) places words into an output buffer called OBUFF. PUTWORD will write the output buffer to the BO file when it is full or when an end of object has been found in the input buffer. Figure 7 illustrates the binary produced by the loader.

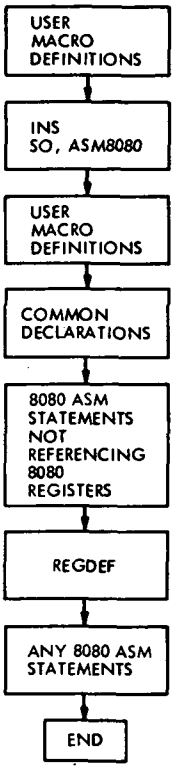


Fig. 1. Generalized 8080 source format

ORIGINAL PAGE IS
OF POOR QUALITY

```
MODCOMP MACRO ASSEMBLY (X)#H  DATE  STANDARD-O/S
  SAMPLE 8080 PROGRAM


  2                         *
  3                         *   THIS IS A SAMPLE INTEL 8080
  4                         *   ASSEMBLY  LANGUAGE PROGRAM
  5                         *
  6                               PGM   SAMPLE
  7                         STOP  MAC
  8                               HLT
  9                               EMP
 10                         LOOP  MAC
 11                               NOP
 12                               JNC   $-2
 13                               EMP
 14                               INS SO,ASM8080
316                         COM   COM   100
317          C 0000         COM1  CEQ   COM
318          C 0001         COM2  CEQ   COM+1
319          C 0002         COM3  CEQ   COM+2
320                               REGDEF
321          A 0007         A     EQU   7
322          A 0000         B     EQU   0
323          A 0001         C     EQU   1
324          A 0002         D     EQU   2
325          A 0003         E     EQU   3
326          A 0004         H     EQU   4
327          A 0005         L     EQU   5
328          A 0006         M     EQU   6
329          A 0000         BC    EQU   0
330          A 0001         DE    EQU   1
331          A 0002         HL    EQU   2
332          A 0003         SP    EQU   3
333          A 0003         PSW   EQU   3
334          A 0500               ORG   #500
335    0500  A 0019               RES   10,25
336                               EXT   SUBR
337                         START CALL  SUBR
338    050A  A 00CD               DFC   #CD
339    050B  A FFFF               DFC   #FFFF,SUBR
       050C  X 0001
340                               JNC   $
341    050D  A 00D2               DFC   #D2
342    050E  A FFFF               DFC   #FFFF,$
       050F  R 050E
343                               MOV   A,B
344    0510  A 0078               DFC   8*A+B+#40
345                               XCHG
346    0511  A 00EB               DFC   #EB
347                               LOOP
348    0512  A 0000               DFC   0
349    0513  A 00D2               DFC   #D2
350    0514  A FFFF               DFC   #FFFF,$-2
       0515  R 0512
351                         LABLE NOP
352    0516  A 0000               DFC   0
353                               JC LABLE
354    0517  A 00DA               DFC   #DA
355    0518  A FFFF               DFC   #FFFF,LABLE
       0519  R 0516
356                               JZ    #18,#60
357    051A  A 00CA               DFC   #CA
358    051B  A 0018               DFC   #18,#60
       051C  A 0060
359                               STOP
360    051D  A 0076               DFC   #76
361          A 0550               ORG   #550
362                               LDA   COM1
363    0550  A 003A               DFC   #3A
364    0551  A FFFF               DFC   #FFFF,COM1
       0552  C 0000
365                               PUSH  PSW
366    0553  A 00F5               DFC   16*PSW+#C5
367                               PUSH  BC
368    0554  A 00C5               DFC   16*BC+#C5
369    0555  A 5448         TEXT1 DFC   "THIS IS SAMPLE TEXT"
       0556  A 4953
       0557  A 2049
       0558  A 5320
       0559  A 5341
       055A  A 4D50
       055B  A 4C45
       055C  A 2054
       055D  A 4558
       055E  A 5420
370    055F  A 534F         TEXT2 DFC   "SO IS THIS"
       0560  A 2049
       0561  A 5320
       0562  A 5448
       0563  A 4953
371    0564  R 050A               END   START
$$
```

Fig. 2. Sample assembly

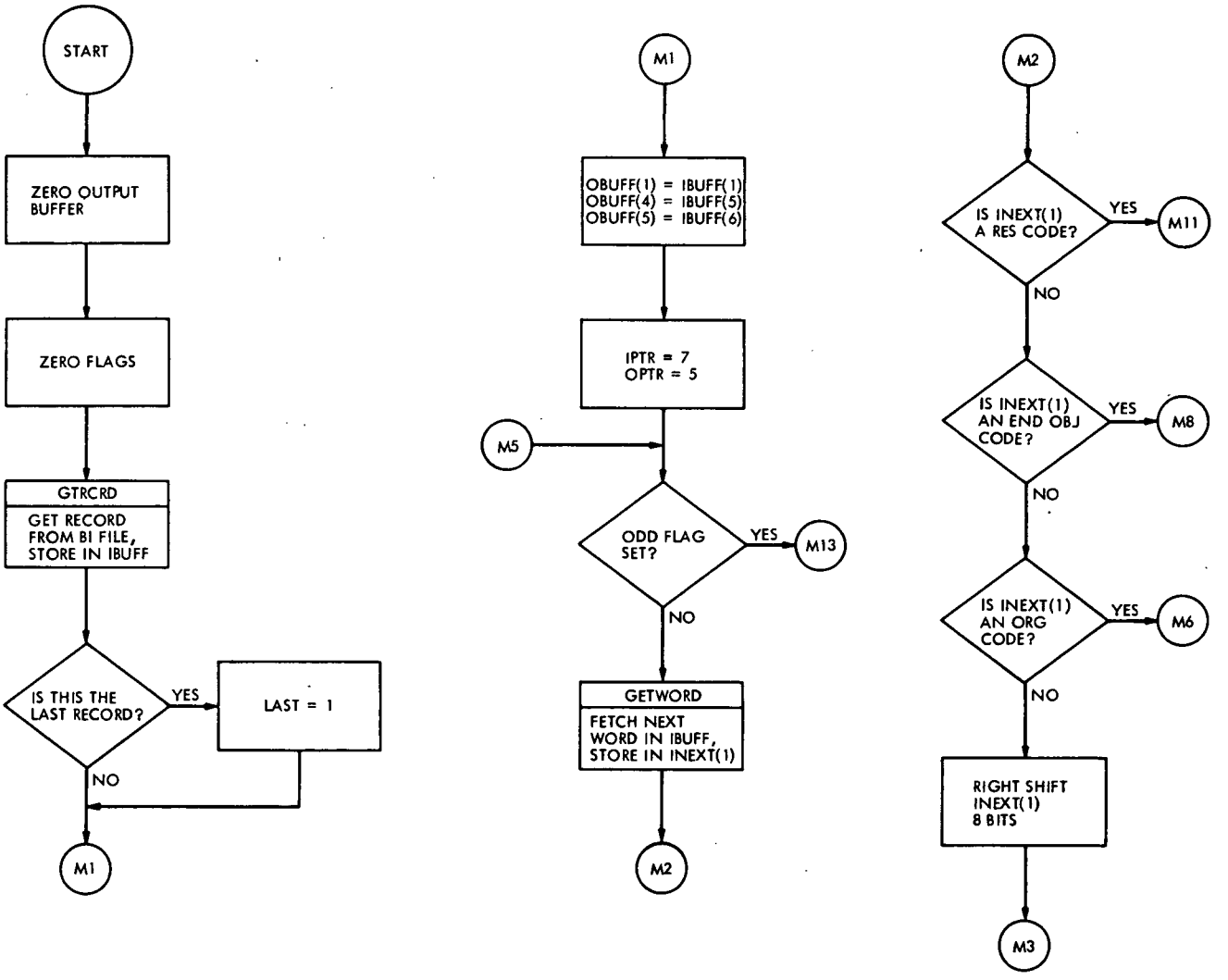


Fig. 3. 8080 loader main program

ORIGINAL PAGE IS
OF POOR QUALITY

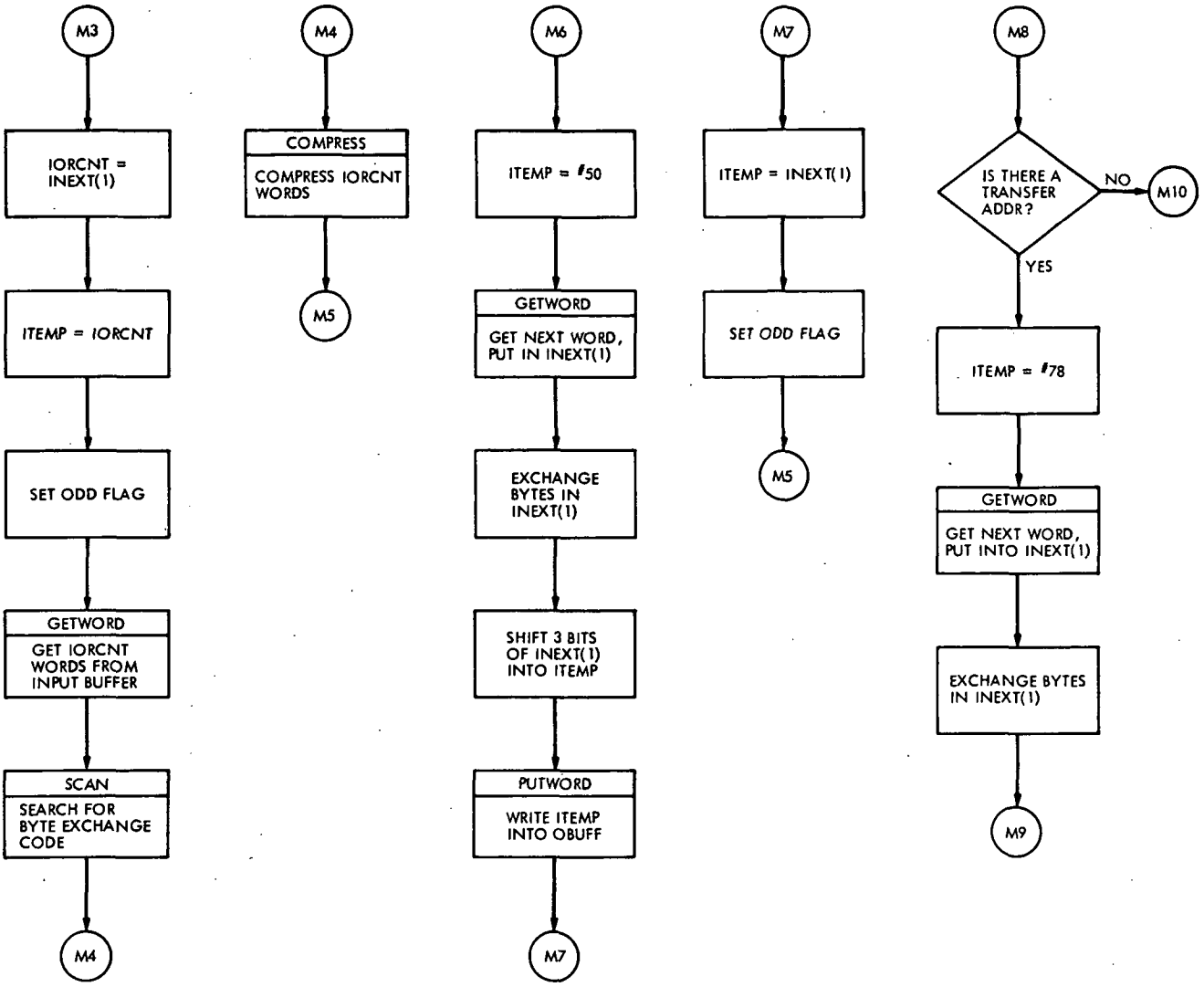
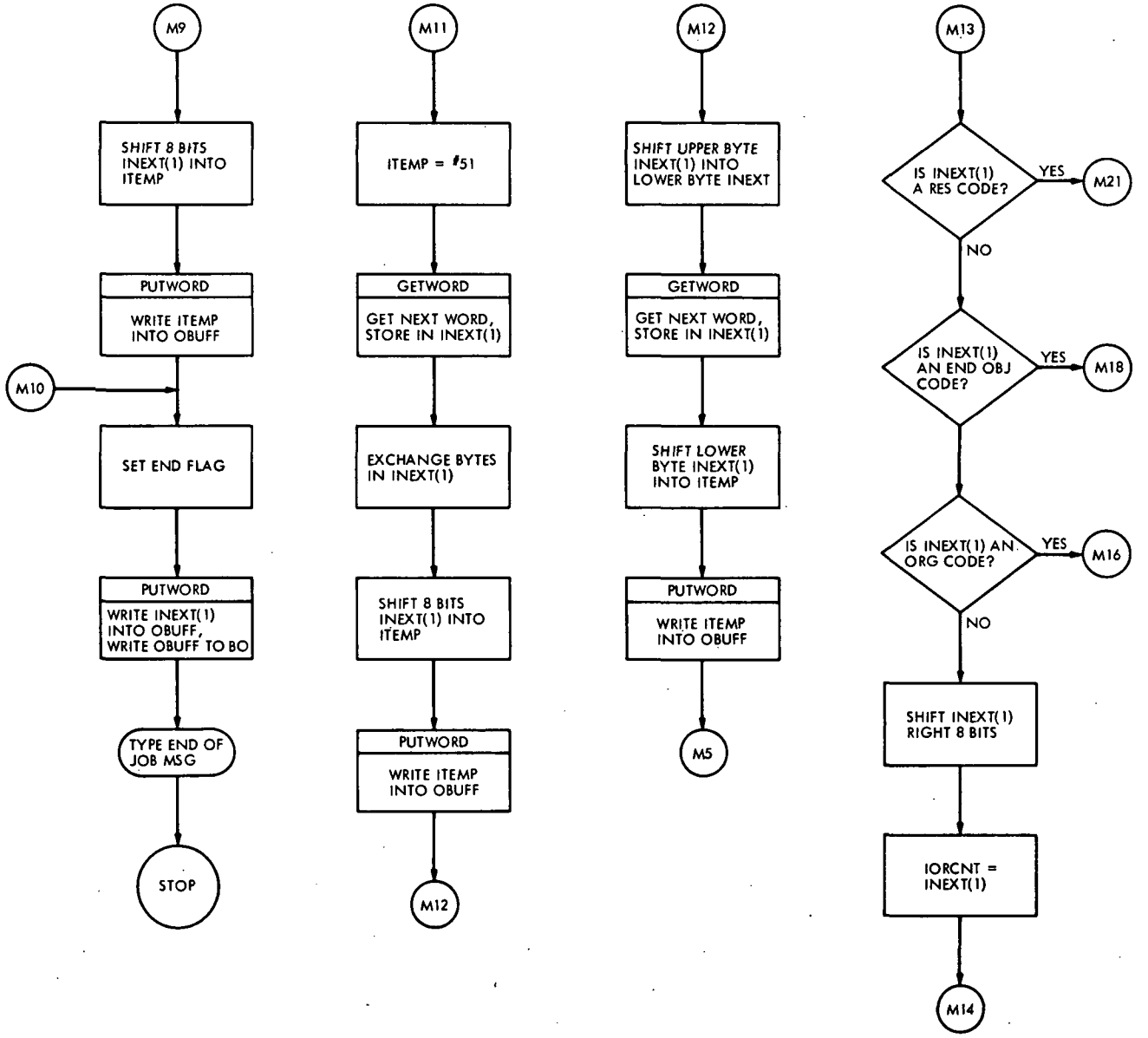

M3
IORCNT = INEXT(1)
ITEMP = IORCNT
SET ODD FLAG
GETWORD
GET IORCNT WORDS FROM INPUT BUFFER
SCAN
SEARCH FOR BYTE EXCHANGE CODE
M4
M4
COMPRESS
COMPRESS IORCNT WORDS
M5
M6
ITEMP = #50
GETWORD
GET NEXT WORD, PUT IN INEXT(1)
EXCHANGE BYTES IN INEXT(1)
SHIFT 3 BITS OF INEXT(1) INTO ITEMP
PUTWORD
WRITE ITEMP INTO OBUFF
M7
M7
ITEMP = INEXT(1)
SET ODD FLAG
M5
M8
IS THERE A TRANSFER ADDR?
NO
M10
YES
ITEMP = #78
GETWORD
GET NEXT WORD, PUT INTO INEXT(1)
EXCHANGE BYTES IN INEXT(1)
M9

Fig. 3 (contd)



ORIGINAL PAGE IS
OF POOR QUALITY

Fig. 3 (contd)

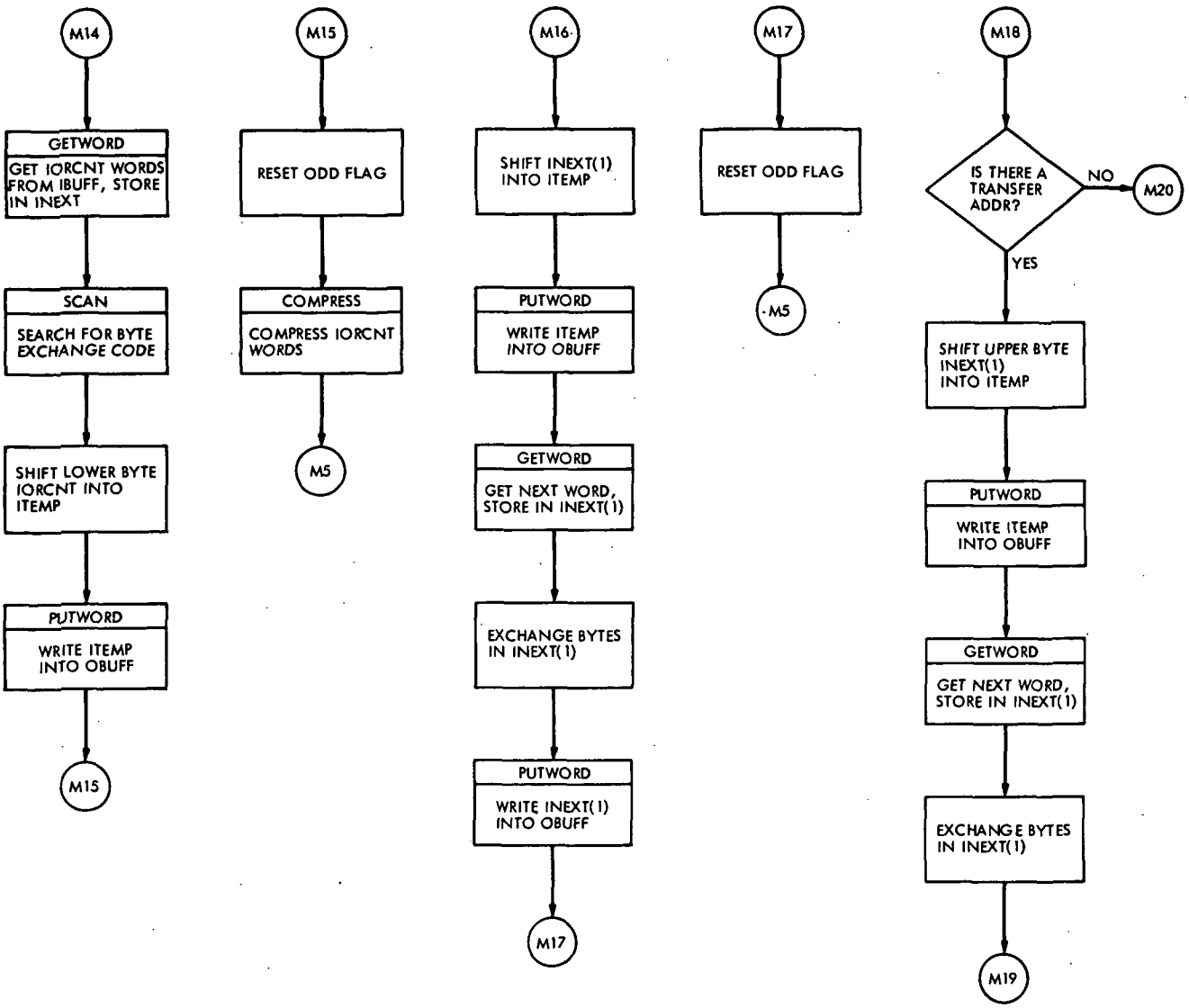

M14
GETWORD
GET IORCNT WORDS FROM IBUFF, STORE IN INEXT
SCAN
SEARCH FOR BYTE EXCHANGE CODE
SHIFT LOWER BYTE IORCNT INTO ITEMP
PUTWORD
WRITE ITEMP INTO OBUFF
M15
M15
RESET ODD FLAG
COMPRESS
COMPRESS IORCNT WORDS
M5
M16
SHIFT INEXT(1) INTO ITEMP
PUTWORD
WRITE ITEMP INTO OBUFF
GETWORD
GET NEXT WORD, STORE IN INEXT(1)
EXCHANGE BYTES IN INEXT(1)
PUTWORD
WRITE INEXT(1) INTO OBUFF
M17
M17
RESET ODD FLAG
M5
M18
IS THERE A TRANSFER ADDR?
NO
M20
YES
SHIFT UPPER BYTE INEXT(1) INTO ITEMP
PUTWORD
WRITE ITEMP INTO OBUFF
GETWORD
GET NEXT WORD, STORE IN INEXT(1)
EXCHANGE BYTES IN INEXT(1)
M19

Fig. 3 (contd)

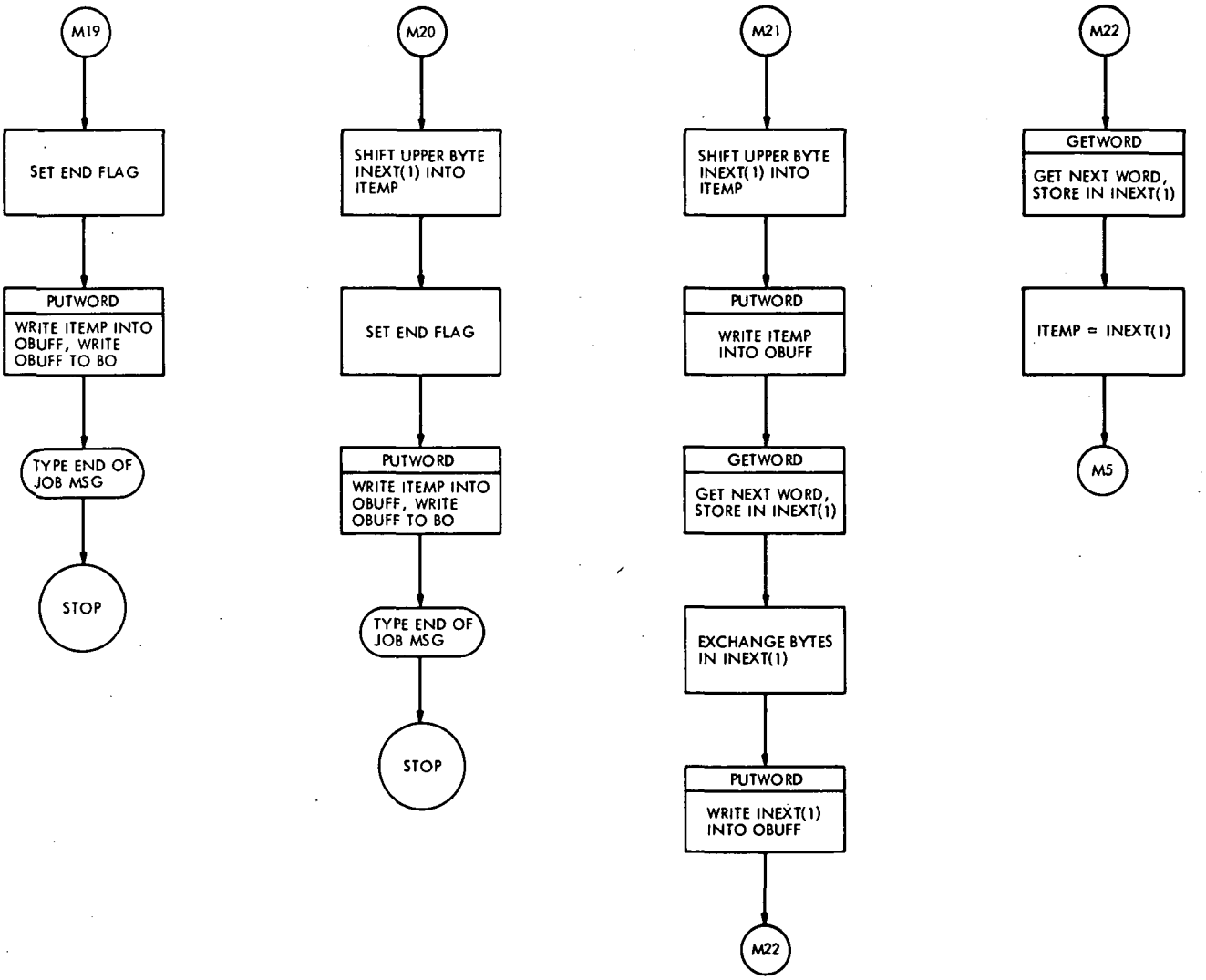


Fig. 3 (contd)

ORIGINAL PAGE IS
OF POOR QUALITY

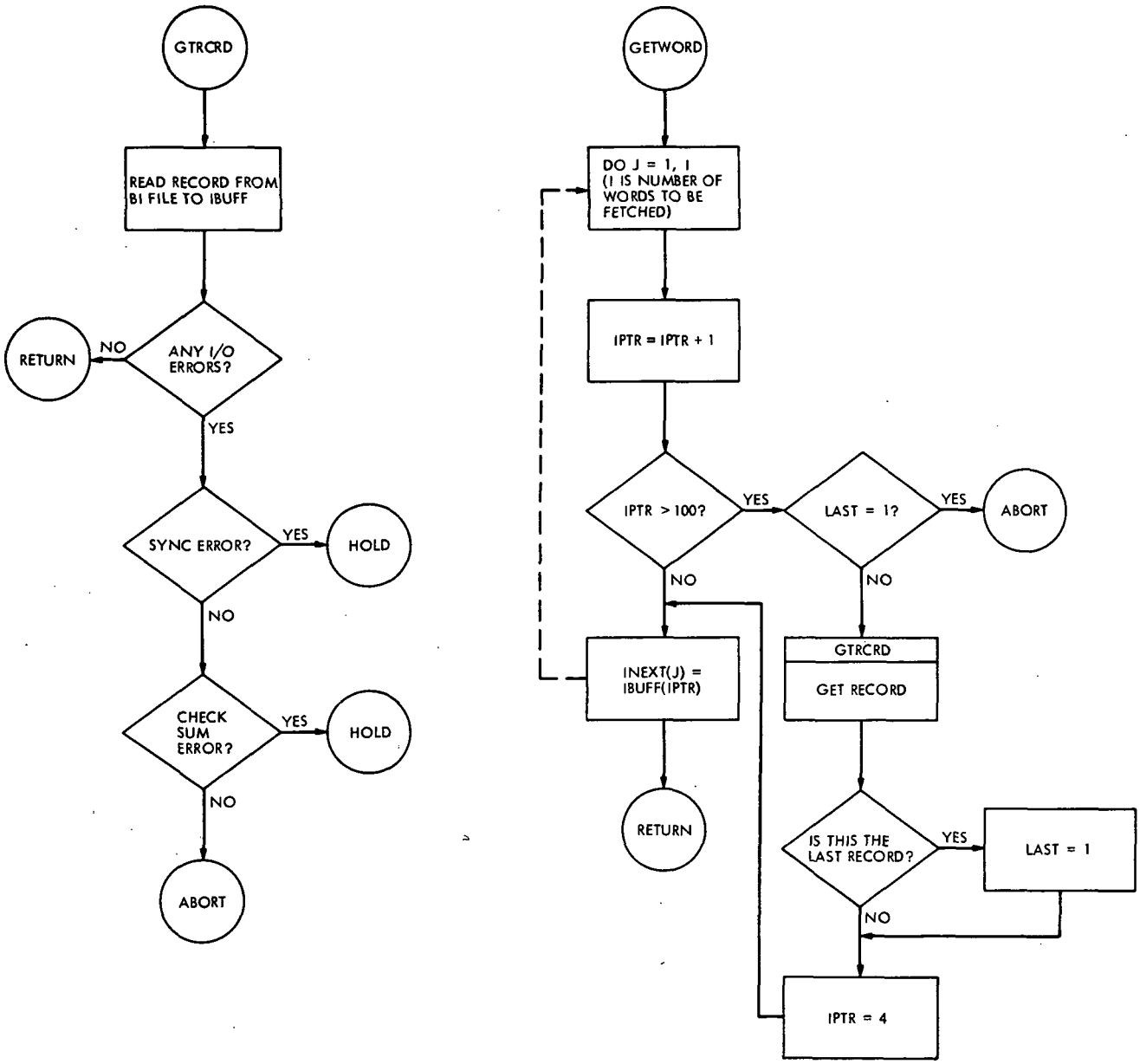


Fig. 4. 8080 loader subroutine GTRCRD, subroutine GETWORD

ORIGINAL PAGE IS
OF POOR QUALITY

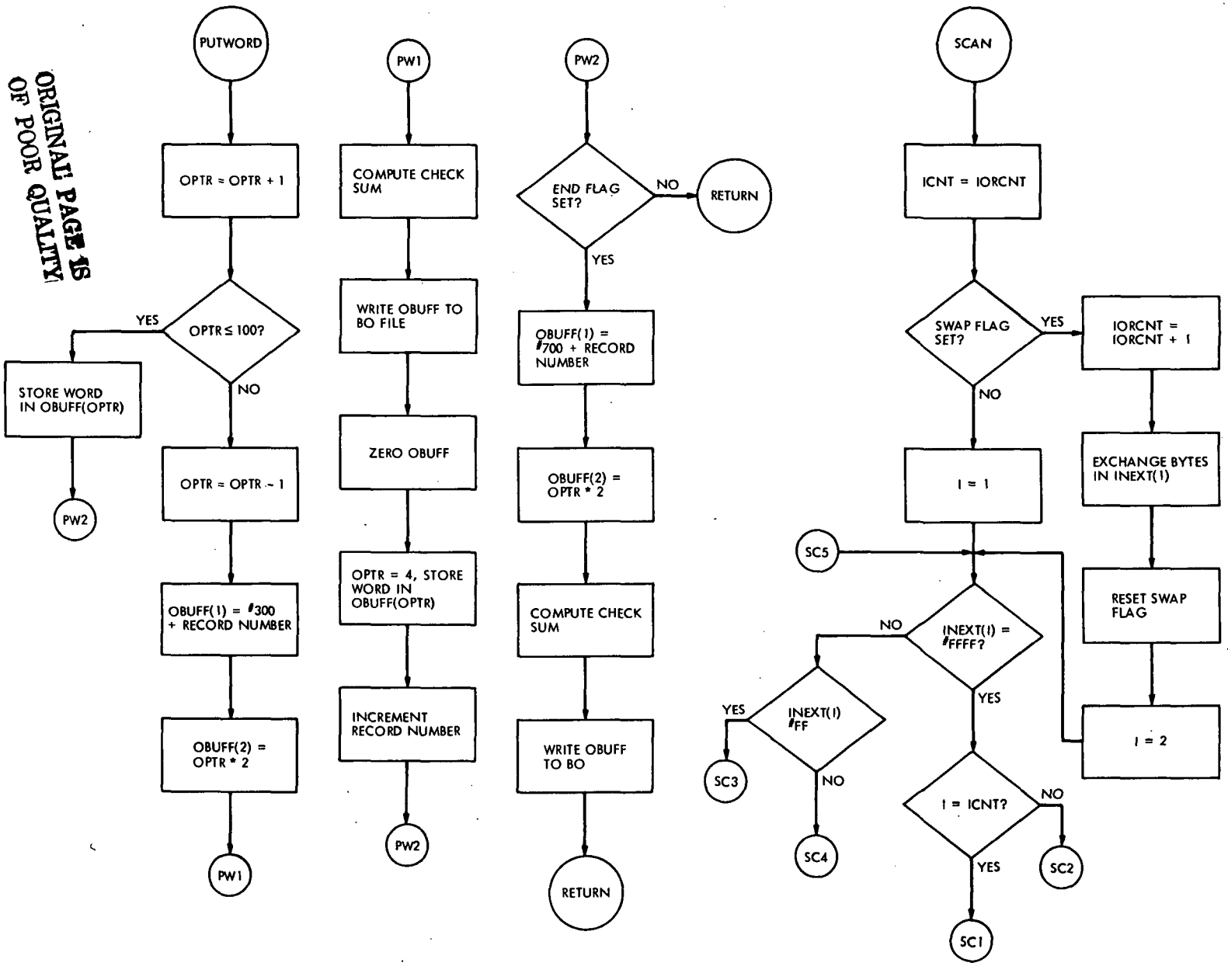


Fig. 5. 8080 loader subroutine PUTWORD, subroutine SCAN

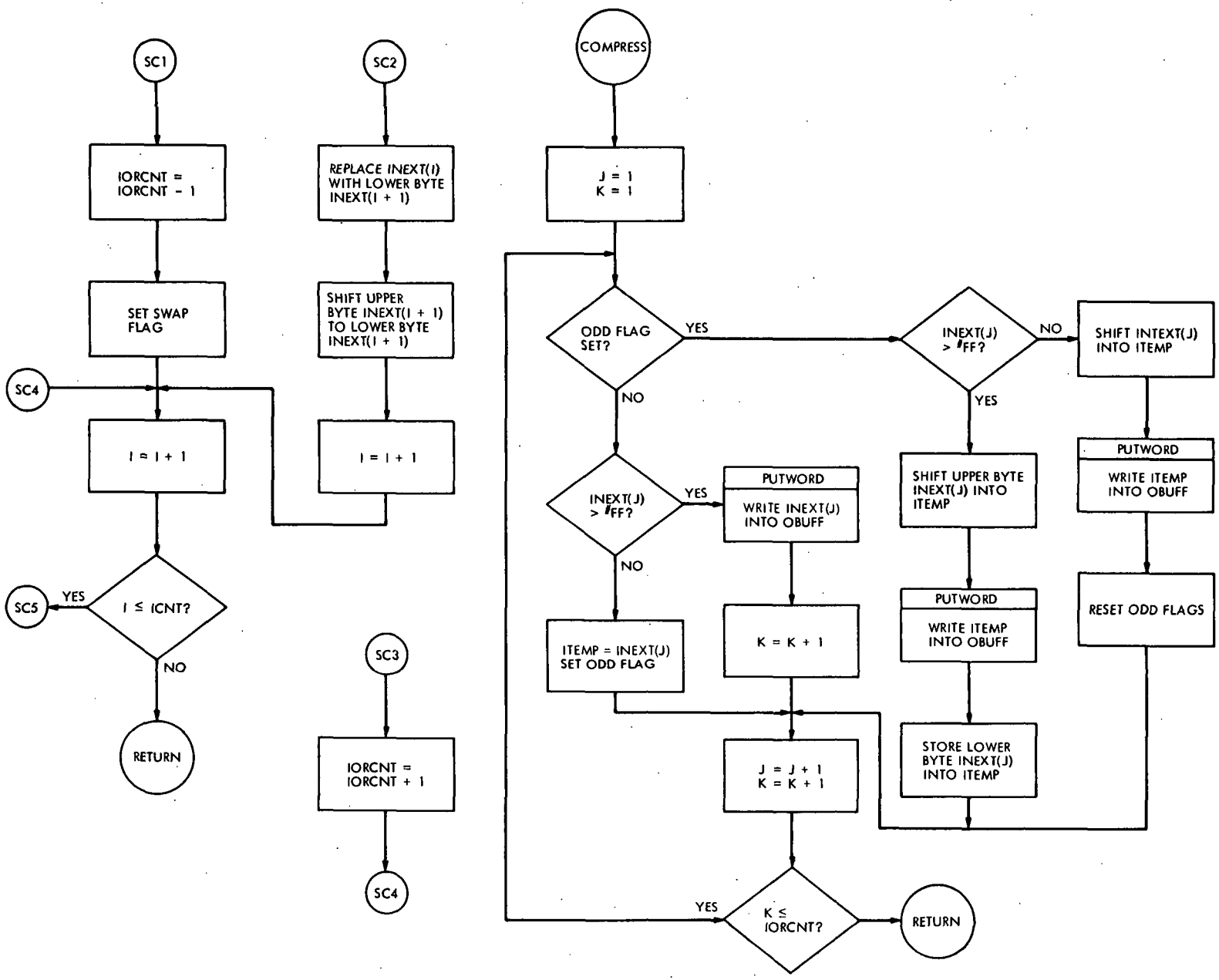
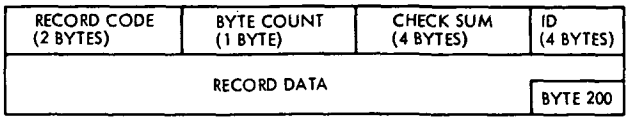
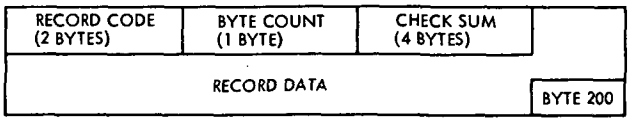


Fig. 6. 8080 loader subroutine SCAN, subroutine COMPRESS

FIRST RECORD



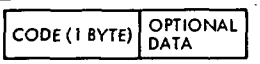
SECOND AND SUBSEQUENT RECORDS



RECORD CODE: $3XX_{16}$ INDICATES XX^{TH}_{16} RECORD

$7XX_{16}$ INDICATES XX^{TH}_{16} RECORD; THIS ONE IS THE LAST RECORD

RECORD DATA



CODES: 51_{16} = RESERVE, DATA CONTAINS RES VALUE

50_{16} = ORIGIN, DATA IS ORIGIN

70_{16} = END OF OBJECT, NO DATA PRESENT

78_{16} = END OF OBJECT, DATA = TRANSFER ADDR

ANYTHING ELSE = BYTE COUNT OF DATA BEFORE NEXT CODE

Fig. 7. Loader format

ORIGINAL PAGE IS OF POOR QUALITY